PRESISTANT: Learning based assistant for data pre-processing

Besim Bilalli[a,b,*], Alberto Abelló[a], Tomàs Aluja-Banet[a], Robert Wrembel[b]

[a] *Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Spain*
[b] *Poznan University of Technology, Poznan, Poland*

Abstract

Data pre-processing is one of the most time consuming and relevant steps in a data analysis process (e.g., classification task). A given data pre-processing operator (e.g., transformation) can have positive, negative, or zero impact on the final result of the analysis. Expert users have the required knowledge to find the right pre-processing operators. However, when it comes to non-experts, they are overwhelmed by the amount of pre-processing operators and it is challenging for them to find operators that would positively impact their analysis (e.g., increase the predictive accuracy of a classifier). Existing solutions either assume that users have expert knowledge, or they recommend pre-processing operators that are only "syntactically" applicable to a dataset, without taking into account their impact on the final analysis. In this work, we aim at providing assistance to non-expert users by recommending data pre-processing operators that are ranked according to their impact on the final analysis. We developed a tool, PRESISTANT, that uses Random Forests to learn the impact of pre-processing operators on the performance (e.g., predictive accuracy) of 5 different classification algorithms, such as Decision Tree (J48), Naive Bayes, PART, Logistic Regression, and Nearest Neighbor (IBk). Extensive evaluations on the recommendations provided by our tool, show that PRESISTANT can effectively help non-experts in order to achieve improved results in their analytic tasks.

Keywords: Data pre-processing, data mining, meta-learning

1. Introduction

Although machine learning algorithms have been around since the 1950s, their initial impact has been insignificant. With the increase of data availability and computing power, machine learning tools and algorithms are making breakthroughs in very diverse areas. Their success has raised the need for mainstreaming the use of machine learning, that is, engaging even non-expert users (i.e., individuals with no proficiency in statistics and machine learning) to perform data analytics. However, the multiple steps involved in the data analytics process render this process challenging.

Data analytics as defined in [1], consists of data selection, data pre-processing, data mining, and evaluation or interpretation. A very important and time consuming step that marks itself out of the rest, is the data pre-processing step. Data pre-processing is challenging but at the same time has a heavy impact on the overall analysis. Specifically, it can have significant impact on the

*Corresponding author
Email addresses: bbilalli@essi.upc.edu (Besim Bilalli), aabello@essi.upc.edu (Alberto Abelló), tomas.aluja@upc.edu (Tomàs Aluja-Banet), robert.wrembel@cs.put.poznan.pl (Robert Wrembel)

© 2019 Elsevier. This manuscript version is made available under the CC-BY-NC-ND 4.0 license http://creativecommons.org/licenses/by-nc-nd/4.0/

generalization performance of a classification algorithm [2, 3], where performance is measured in terms of the ratio of correctly classified instances (i.e., predictive accuracy).

The main tools used for data analysis (i.e., Weka, RapidMiner, Knime, Orange, SAS, IBM SPSS Modeler, scikit-learn, R) overlook data pre-processing when it comes to assisting non-expert users in improving the overall performance of their analysis. These tools are usually meant for professional use, for users who know exactly which pre-processing operators to apply. However, the staggeringly large number of available pre-processing operators (transformations) overwhelm non-experts, and they require support.

To this end, our work focuses on assisting users by reducing the number of pre-processing options to a bunch of potentially relevant ones. The goal is to retain only transformations that have high positive impact on the analysis. Like this, by recommending only a small set of transformations, we aim at reducing the time consumed in data pre-processing and at the same time, retaining only the relevant transformations, we aim at improving the final results of the analysis. As a case study, because of their use in practice, we focus on classification problems. Therefore, our method can be used to recommend pre-processing operators that improve the performance of a selected classification algorithm (i.e., increase algorithms predictive accuracy in a given classification problem). However, note that our method can be easily extended to deal with regression problems too.

Contributions. The main contributions of this paper can be summarized as follows:

- We apply meta-learning techniques to develop a system that is capable of recommending pre-processing operators (transformations) that positively impact the final result of some classification tasks. Our method is based on training an algorithm to learn the impact of pre-processing operators and then use it to predict and ultimately rank different pre-processing operators.

- We perform an extensive experimental evaluation to compute the accuracy of the rankings with regards to a) the whole set of transformations and b) the top-K. For the former, we obtain an accuracy of 61% as an average for all the algorithms we consider. For the latter (i.e., K=1), the accuracy increases up to 68% on average.

- We evaluate our rankings with regards to the benefit or gain obtained from the user's point of view and we measure it using a classical information retrieval metric, Discounted Cumulative Gain (DCG). For the whole set of transformations we are as close as 73% to the gain obtained from the ideal rankings, whereas for the top-1 we are as close as 79%.

- We perform an empirical study comparing the ability of real users against our approach on finding a transformation that positively impacts the classification accuracy, on a set of randomly selected classification problems. Our approach, on average, performs 2.5 times better.

The remainder of this paper is organized as follows. In Section 2, we give an overview on data pre-processing and perform an empirical study on the impact of pre-processing. In Section 3, we report on what has been done so far in the line of automating the data pre-processing step, thus the related work. In Section 4, we provide background information on meta-learning and its use on learning the relationship between pre-processing operators and data mining algorithms. In Section 5, we present our tool — PRESISTANT, and its core functionalities. In Section 6, we provide

Transformation	Technique	Attributes	Input Type	Output Type
Discretization	Supervised	Local	Continuous	Categorical
Discretization	Unsupervised	Local	Continuous	Categorical
Nominal to Binary	Supervised	Global	Categorical	Continuous
Nominal to Binary	Unsupervised	Local	Categorical	Continuous
Normalization	Unsupervised	Global	Continuous	Continuous
Standardization	Unsupervised	Global	Continuous	Continuous
Replace Miss. Val.	Unsupervised	Global	Continuous	Continuous
Replace Miss. Val.	Unsupervised	Global	Categorical	Categorical
Principal Components	Unsupervised	Global	Continuous	Continuous

Table 1: List of transformations (data pre-processing operators)

an extensive evaluation of our approach, and finally in Section 7, we provide the conclusions and discuss some ideas for future work.

2. Data pre-processing

Data pre-processing consumes 50-80% of data analytics time [4]. The reason is because data pre-processing encompasses a broad range of activities. Sometimes data needs to be transformed in order to fit the input requirements of the machine learning algorithm (e.g., if the algorithm accepts only data of numeric type, data is transformed accordingly). Sometimes, data requires to be transformed from one representation to another e.g., from an image (pixel) representation to a matrix (feature) representation [5], or data may even require to be integrated with other data to be suitable for exploration and analysis [6]. Finally, and more importantly, data may need to be transformed with the only goal of improving the performance of a machine learning algorithm [7]. The first two types of transformations are more of a necessity, whereas the latter is more of a choice, and since an abundant number of choices exist, it is time consuming to find the right one. In this work, we target the latter type of pre-processing, and as such, the transformations taken into consideration are of the type that can impact the performance of data mining algorithms (i.e., classification algorithms). In our experiments, we consider the transformations that are used the most in Weka [8]. The types of transformations and their characteristics are listed in Table 1.

In Table 1, a transformation is described in terms of: 1) the *Technique* it uses, which can be `Supervised` — these transformations consider the class of the value when applied, and `Unsupervised` — when the class of the value is not considered and the transformation is applied only considering the attribute being transformed, 2) the *Attributes* it uses, which can be `Global` — the transformation is applied to all compatible attributes at once, and `Local` — the transformation is applied to selected attributes, 3) the *Input Type*, which denotes the compatible attribute type for a given transformation, and it can be `Continuous` — when it represents measurements on some continuous scale, or `Categorical` — when it represents information about some categorical or discrete characteristics, 4) the *Output Type*, which denotes the type of the attribute after the transformation and it can similarly be `Continuous` or `Categorical`. A short description for each category of transformations from Table 1 follows.

Discretization - the process of converting or partitioning continuous attributes to discretized or nominal/categorical attributes.

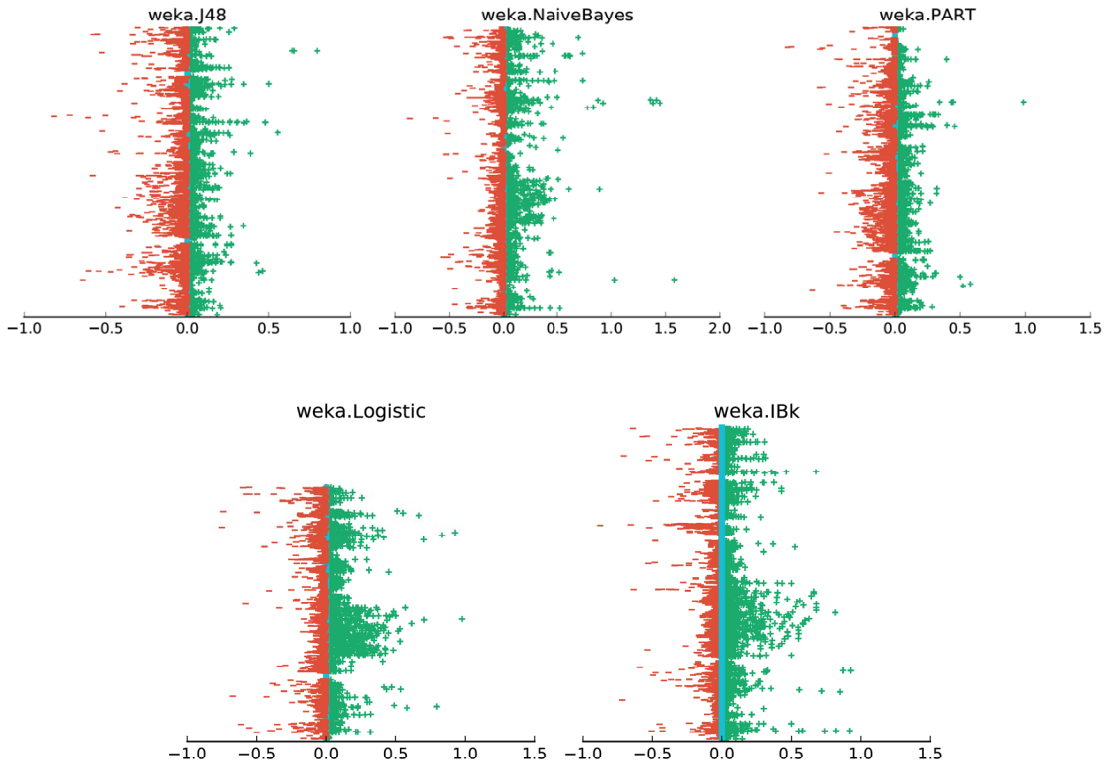

Figure 1: Distributions of the (relative) impact induced by transformations on the selected algorithms

Nominal to Binary - the process of converting nominal/categorical attributes into binary numeric attributes.

Normalization - the process of normalizing numeric attributes such that their values fall in the same range.

Standardization - the process of standardizing numeric attributes so that they are in the same scale.

Missing Value Imputation - the process of imputing missing values.

Principal Component Analysis - linear dimensionality reduction technique. The goal is to reduce the large number of directly observable features into a smaller set of indirectly observable features.

Note that the list of transformations considered is not exhaustive and thus many other transformations can be added (e.g., Outlier Detection).

2.1. Empirical analysis on the overall impact of data pre-processing

Other than theoretical analysis [3], to the best of our knowledge, there is not much work on empirically studying the impact of pre-processing operators on real world classification problems. Therefore, in order to assess the impact of pre-processing, we randomly selected 533 datasets[1] from

[1]See `http://www.essi.upc.edu/~bbilalli/presistant.html#datasets` for a list of datasets and their characterstics.

Category	Algorithm	Default Configuration	Description
Tree	Decision Tree (J48)	-C 0.25 -M 2	-C Confidence value -M Minimum number of instances in the two most popular branches
Bayes	Naive Bayes	-K False -D False -O False	-K use kernel density estimator -D use supervised discretization -O display model in old format
Rules	PART	-C 0.25 -M 2 -Q 1 -R False -N 3 -B False 3 -U 1	-Q seed for random data shuffling -R use random error pruning -N number of folds -B use binary splits -U generate unpruned dec. list
Function	Logistic Regression	-R 1.0E-8 -M until convergence	-R set ridge in log-likelihood -M maximum number of iterations
Lazy	Nearest Neighbor (IBk)	-K 1 -F False -E False -X False -A LinearNNSearch	-K number of nearest neighbors -F weight neighbors by their distance -E minimise mean squared error -X use cross validation -A nearest neighbor search algorithm

Table 2: List of ML algorithms

the OpenML [9] repository and applied the pre-processing operators shown in Table 1. The datasets used in our experiments are: 34% of *Continuous* type — containing only numeric attributes, 11% of *Categorical* type – containing only categorical attributes, and 55% of *Combined* type — containing both categorical and numeric attributes.

Finally, we used 5 different classification algorithms (i.e., J48, Naive Bayes, PART, Logistic and IBk)[2] — cf. Table 2 for their default configurations, and measured their performance (e.g., predictive accuracy) on datasets before and after the transformations were applied. In Figure 1, we show the scatter plots of the relative change in predictive accuracy (evaluated with 10-fold cross-validation). In each scatter plot, we visualize the impact of all the transformations for a given algorithm, where, *red*, *blue*, and *green*, denote *negative*, *zero*, and *positive* impact, respectively. Moreover, each point represents a transformation applied to a dataset and a different dataset is represented along every horizontal line. The total number of individual transformations applied — hence the number of points, to all the datasets is approximately 25,000. This number comes as a result of the fact that transformations are applied depending on whether they are Local or Global (as classified in Table 1). If a transformation is of type Global it is applied only once to the set of all compatible attributes (e.g., normalizing all numeric attributes), whereas if it is Local, it is applied to: 1) every compatible attribute separately (e.g., discretizing one attribute at a time), and 2) all the set of compatible attributes (e.g., replacing missing values of all attributes).

Furthermore, the application of transformations depends on the attribute types of datasets. Therefore, certain transformations cannot be applied to certain datasets because of the mismatch with respect to the expected types of attributes (e.g., Discretization cannot be applied to a dataset with only Categorical attributes).

Finally, out of the total number of transformations applied per algorithm, 7% are of type Global and the rest are of type Local. This is rather expected because a Global transformation is applied

[2]We chose one representative algorithm for 5 different classes of classification algorithms in Weka.

Figure 2: The overall impact of transformations expressed in percentage

only once — to all the compatible attributes, whereas a Local transformation can be applied many times — to different combinations of attributes.

Observing Figure 1, one may conclude that:

- overall, transformations impact the final result of the analysis (i.e., they impact the predictive accuracy of the classification algorithms considered),
- the magnitude of the impact is heterogeneous, and
- there is no clear winner when it comes to the sign of the impact, i.e., transformations do not always impact positively or they do not always impact negatively.

To confirm the latter, in Figure 2, we show the percentages of the positive, negative and neutral impacts using bar plots.

Figure 2 shows that transformations are almost uniformly distributed with respect to the sign of impact, which means that it may be challenging to distinguish among transformations that affect positively or negatively the final result.

2.2. Empirical study on the impact, per pre-processing operator

In the previous section, we argued that in general, the impact of pre-processing is sound, but it may be difficult to find or predict the transformations that have positive impact. The analysis was performed on all transformations without distinguishing their types. The point now, is to check whether the previous conclusions still hold when we delve into studying categories of transformations separately (e.g., discretization), or conversely to the general picture, there exist some patterns (e.g., discretization has mainly positive impact).

In Figure 3, in a matrix like structure, we show the impact of every transformation from Table 1, for every algorithm considered. Circles are sized by the distance from a perfectly uniform distribution of the impact (i.e., 33% negative, 33% zero, 33% positive), and they are colored by the winner sign (i.e., red if negative, blue if zero, and green if positive impact is the winner). Thus, the bigger the circle and the sharper the color, the more obvious the pattern for a given transformation.

For the sake of illustration let us consider a real execution. Nearest Neighbor for Normalization has a distribution of $10\%, 80\%, 10\%$ for negative, zero, and positive impact, respectively, and NaiveBayes for the same transformation has a distribution of $25\%, 30\%, 45\%$. Then, the sizes of the circles are determined by the euclidean distance between $(10, 80, 10)$ and $(33, 33, 33)$ for the first algorithm, and the distance between $(25, 30, 45)$ and $(33, 33, 33)$ for the second algorithm. The distance for the first algorithm (57.15) is obviously higher than the distance for the second algorithm (14.73), and hence the size of the circle. Furthermore, to define the colors of the circles, the distributions for negative, positive and zero impact, participate proportionally to the RGB (red, green, blue) coloring scheme. Hence, for the above mentioned example for the first algorithm,

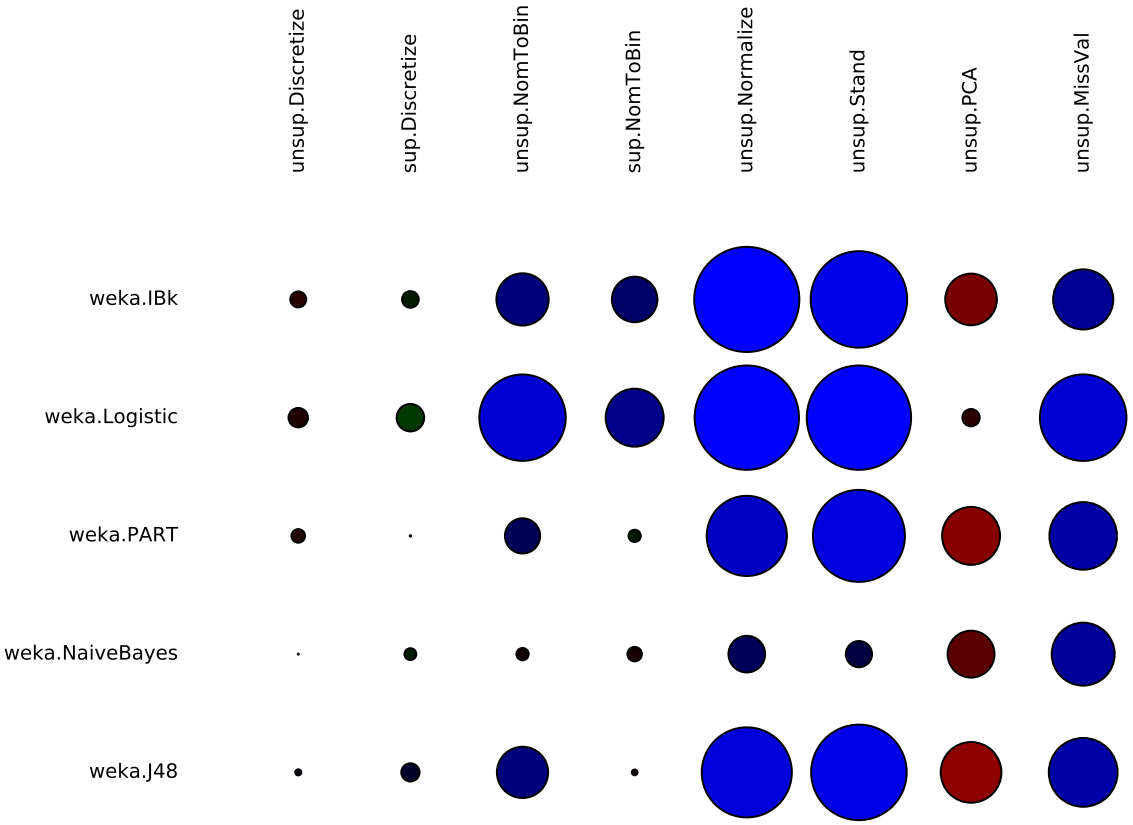

Figure 3: Impact of pre-processing operators

color blue will be more decisive, and for the second, green. Yet, for the first algorithm, the color will be sharper than for the second, because of the values being higher.

The patterns emerging from the plot may help in two directions. First, they can be used to devise basic rules or heuristics, i.e., if a transformation has a big blue circle for a given algorithm, then the transformation can be discarded because it is basically of no use for that particular algorithm — since most of the time it does not affect the final result. Secondly, the patterns reveal the more difficult transformations with respect to finding the impact, i.e., if a transformation has a small circle and no clear color for a particular algorithm, it means that the distribution of the impact is close to uniform and hence a simple rule may not help in finding the impact of the transformation. The latter rises the need for developing more sophisticated techniques. To this end, we propose to learn the relationship between data pre-processing operators and data mining algorithms using meta-learning, and we delve into its details in the next section.

2.2.1. Simple heuristics/rules as result of the empirical study

In Figure 3, circles of bigger size give clear patterns for devising simple heuristics. Notice that the bigger circles are usually of blue color. This means that the transformations for the corresponding algorithms are not of much use, since they do not impact the performance of the algorithms

on the tested datasets. Some of the blue circles are obviously expected. For instance, it is well known that Normalization and Standardization do not impact the performance of Decision Trees (i.e., J48). Hence, a simple heuristic would be that, when a Decision Tree is chosen, Normalization and Standardization should not appear in the palette of possible transformations. The same holds for Logistic Regression. These transformations do not impact its performance.

However, a counter-intuitive pattern appearing is that of Normalization and Standardization with Nearest Neighbor. One would expect an impact of transformations, yet the circles are big and the color is blue, implying no impact. We checked the actual Weka implementation in order to understand why was this happening. It resulted that Nearest Neighbor in Weka, by default uses a normalized Distance algorithm. Hence, Normalization is implicitly performed inside the learning algorithm and as a consequence, an external Normalization does not impact the performance. Similarly, for Logistic Regression we realized that NominalToBinary and Missing Value Imputation are applied implicitly in Weka. Mind that this kind of implicit default configurations may lead to unexpected results and as noted in [10] may have significant implications for both practitioners and researchers.

The rest of the transformations in Figure 3 have smaller circles, and this implies that they have impact on the performance of algorithms. However, in the case of PCA for instance, although smaller, the circles are reddish. This indicates that although PCA impacts the performance, it needs to be carefully applied. An agnostic application of PCA may lead to a negative impact on the overall performance, which indicates that PCA may need to be used by more experienced users.

3. Related work

A lot of ongoing research aims at addressing the problem of providing user assistance for the different steps of knowledge discovery[3]. In general there is a tendency to develop (semi) automatic systems that provide user assistance in one or many steps altogether. At the earliest stage the focus has been to provide support exclusively for the data mining step. Recently however, the direction has shifted towards designing systems that specifically provide user assistance in the data pre-processing step and also systems that aim at fully automating the knowledge discovery process (i.e., automatically generate data analytics flows). Therefore, these works can be grouped into 3 broad categories: support for data pre-processing, support for data mining, and support for data analytics.

User support for data pre-processing. As defined in [11], data pre-processing is typically performed either to repair data (e.g., based on some quality rules), or to transform data such that it yields better results in the later stage of statistical analysis (e.g., when applying a machine learning algorithm on top of it). To provide a classification of different systems with regard to the user support for pre-processing, we followed an approach similar to [11]. That is, we classified different systems under the questions of *What*, *How*, and *Why*. Under *What*, we are interested on what pre-processing task [12] a system supports, and this can be *cleaning*, *preparation*, *curation*, or *wrangling*, and what kind of support is provided, whether *automatic* (i.e., the user does not interact with the system) or *interactive* (i.e., the user plays a key role in the process). Next, under *How*, we seek an answer to the question of how the system manages to provide such a support,

[3]It is interchangeably refer to as *data analytics*

	What						How															Why
	Task				**Supp.**		**Method**							**Input**								
System	Cleaning	Preparation	Curation	Wrangling	Automatic	Interactive	Learning	Optimization	Ad-hoc Technique	Program. by Demons.	Rule based method	Program. by Example	Decision Theory	Quality Rules	Metadata	Knowledge Base	User Interaction	Input-Output	Web Tables	ML Algorithm	Crowdsourcing	Impact on Analysis
Wrangler [14]	✓	✓		✓		✓				✓					✓	✓	✓					
GDR [15]	✓					✓	✓						✓	✓			✓					
SST [16]				✓		✓						✓						✓				
Data Tamer [17]	✓	✓	✓			✓	✓								✓	✓					✓	
NADEEF [18]	✓				✓				✓		✓			✓	✓							
Llunatic [19]	✓				✓				✓					✓			✓					
CDC [20]	✓					✓	✓								✓		✓					
BigDansing [21]	✓				✓			✓	✓		✓			✓		✓						
KATARA [22]	✓					✓			✓							✓					✓	
ActiveClean [23]	✓	✓				✓			✓		✓			✓						✓		✓
DataXFormer [24]	✓	✓		✓		✓			✓							✓		✓	✓			
Foofah [25]	✓	✓		✓	✓						✓	✓						✓				
Learn2Clean [26]	✓	✓			✓		✓	✓												✓		✓
DPD [27]		✓			✓		✓	✓												✓		✓

Table 3: Classification of the related work

that is the method and the input used by the system. Finally, in *Why* we have two options only, whether the support provided is intended to impact the analysis (i.e., application of a machine learning algorithm on top of the pre-processed data) to be performed at a later stage, or not.

In Table 3, we provide a classification of the related work ordered by year of publication. To find the systems for comparison, we first started with Wrangler [13], which is one of the most influential works in this area and then for each following year we selected one or more other influential works. As a measure of influence, we consider the number of citations for a given work.

As observed from the table, most of the systems provide user support for cleaning tasks. However in terms of operators used, the intersection of different pre-processing tasks is not an empty set. For instance, sometimes a wrangling operator (e.g., string manipulation), can be used both for wrangling or cleaning. For more details about the different pre-processing tasks the reader is referred to [12].

Furthermore, as it can be read from the table, there are systems that discover patterns and detect errors in data and then automatically infer relevant transformations. For instance, in Wrangler [14], Foofah [25], and SST [16], relevant transformations are learned either by demonstration or by example, the difference being that in the former the user directly manipulates the visualized data and in the latter she needs to provide the output (to be) data. A clear requirement here is that the user needs to know the final shape of the data.

Next, systems like GDR [15], NADEEF [18], Llunatic [19], and BigDansing [21] (semi) automate the detection and repairing of violations with respect to a set of heterogeneous and ad-hoc constraints. Many types of quality constraints like functional dependencies, conditional functional dependencies, multivalue dependencies, and ETL rules can be defined. Their goal is to cope with multiple queries holistically and optimize their application, considering as well the implications of a Big Data setting [21]. Yet, a clear requirement of these systems is that the quality rules need to be defined in advance.

KATARA [22] and DataXFormer [24] also infer transformations, however, this time using external knowledge stored in knowledge bases, web tables, or even by knowledge obtained interacting with crowds.

In DataTamer [17], they deal with the end to end curation (e.g., integration, de-duplication) of data from different sources and a learning algorithm is used to learn the transformations applied by the user such that it can recommend transformations at later stages. Likewise in CDC [20], an algorithm is trained to learn the evolution of the data and its semantics in order to dynamically suggest repairs based on the accumulated evidence to date.

Note that to guide the user support, none of the aforementioned systems considers the impact of pre-processing on a potential analysis. Clearly pre-processing is considered in isolation from data analysis (i.e., data mining). Only systems like ActiveClean [23], and the more recent ones like Learn2Clean [26] and DPD [27] consider the impact of pre-processing on the final analysis.

In ActiveClean [23], the system aims at prioritizing the cleaning of records that are more likely to affect the results of the statistical modeling problems, assuming that the latter belong to the class of convex loss models (i.e., linear regression and SVMs). Hence, instead of recommending the transformations to be applied, the system recommends the subset of data which needs to be cleaned at a given point. The type of pre-processing to be applied is left to the user, assuming that the user is an expert. Next, based on a reinforcement learning technique, for a given dataset, and an ML model, Learn2Clean [26] selects the optimal sequence of tasks for pre-processing the data such that the quality of the ML model result is maximized. Similarly in DPD [27], a Bayesian Optimization technique is used to demonstrate how it can automatically select and tune pre-processing operators to improve the baseline score with a restricted time budget. The latter two, in terms of the goal and the input used qualify as the most related ones to our approach, however in terms of the technique used to provide user support they differ a lot. That is, after each exploration of pre-processing operators they apply the machine learning algorithms for real and therefore induce an additional cost. Although they intend to optimize the search space and therefore reduce the number of times the ML algorithm is applied, the ML algorithm is still applied on the pre-processed data. In PRESISTANT however, we aim to predict the impact of pre-processing operators without applying the ML algorithm and therefore we do not incur an additional time cost.

Lastly, RapidMiner[4] and IBM SPSS Modeler[5] also provide modules to facilitate the data pre-processing step. However, these modules are based on rule based methods (e.g., if the percentage of missing values is greater than 50% exclude the variable from the subsequent analysis) and therefore are complementary to our approach (cf. Section 5.1). For a better understanding, in Figure 4 we provide a SWOT analysis of our approach.

User support for data mining (model selection) [28]. The main systems for providing support in data mining are dubbed as *Expert systems* and *Meta-learning systems*.

Expert systems [29, 30, 31] are the first and simplest systems to provide help to the user during the data mining phase. Their main component is a knowledge base consisting of expert rules. Given the input, either from the user or extracted from the dataset, the rules are used to determine the mining algorithms to be recommended.

Meta-learning systems (MLS) [32, 33, 34] are more advanced than *Expert systems*. The rules that were statically defined by the experts in the previous category are dynamically learned here.

[4]https://rapidminer.com

[5]https://www.ibm.com/products/spss-modeler

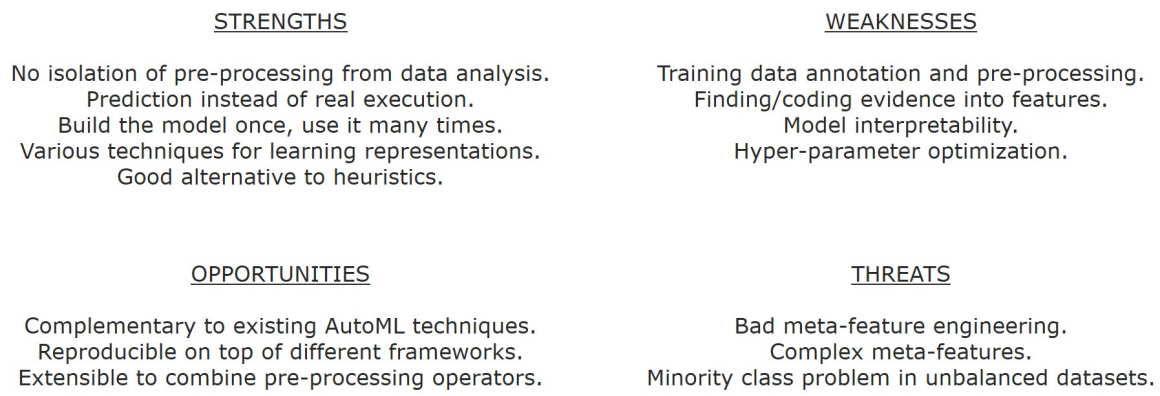

Figure 4: SWOT analysis

MLSs try to discover the relationship between measurable features of the dataset and the performance of different algorithms, which is a standard learning problem. The learned model is then used to predict the most suitable data mining algorithm for a given dataset.

In general, the drawback of these systems is that they overlook the impact of pre-processing.

User support for data analytics (knowledge discovery) [35]. When it comes to automating the whole knowledge discovery process we distinguish between *Case-based reasoning systems*, *Planning-based data analysis systems*, and *AutoML (automated machine learning) systems*.

Case-based reasoning systems (CBS) [29, 36, 37] store the successfully applied workflows (i.e., machine learning pipelines) as cases, in a *case base*, with the goal of reusing them in the future. When faced with a new problem (i.e., dataset) provided by the user, these systems return k similar cases, which can be further adapted to the current problem.

Planning-based data analysis systems (PDAS) [38, 39, 40] are able to autonomously design valid workflows without relying on similarities. To this end, the workflow composition problem is treated as a planning problem, where a plan is built by combining operators that transform the initial problem into accurate models or predictions. In order to construct valid workflows, the input, output, preconditions, and effects (IOPE) of each operator (e.g., pre-processing or data mining algorithm) need to be formally defined. Plenty of workflows are then generated by combining operators that syntactically complement one another.

AutoML systems [41, 42, 43] refer to systems that try to automatically optimize the hyperparameters of operators. The goal is to automatically generate workflows or machine learning pipelines that give optimal results for the task at hand. Typically, Bayesian optimization methods are used to tune and optimize the hyperparameters. Since Bayesian optimization is randomized and it starts from a random configuration of hyperparameters, meta-learning has been used to find a good seed for the search [44].

In summary, full automation of knowledge discovery has been an ultimate goal of many research works. Yet, such an automation has shown to be computationally expensive, mainly due to the search space involved (i.e., pre-processing and mining operators). Therefore, the usability of such approaches in realistic scenarios is rather limited. However regardless of the latter, our approach

of finding a set of relevant transformations can be seen as complementary to these solutions, since it helps in pruning the large search space.

4. Background: meta-learning for predicting the impact of pre-processing

Meta-learning is a general process used for predicting the performance of an algorithm on a given dataset. It is a method that aims at finding relationships between dataset characteristics and data mining algorithms [32]. Given the characteristics of a dataset, a predictive meta-model can be used to foresee the performance of a given data mining algorithm. For instance, in a classification problem, meta-learning can be used to predict the predictive accuracy of a classification algorithm on a given dataset and hence provide user support in the mining step [45].

In other cases, meta-learning has been used to find workflows for the complete data analytics process [46], or it has shown to provide good heuristics for the combined algorithm selection and hyperparameter (CASH) optimization problem [42].

In our previous works, we showed that meta-learning can also be used to provide support in the pre-processing step [47, 48]. This can be done by learning the impact of data pre-processing operators (transformations) on the result of the eventual analysis. This way, meta-learning pushes the user support to the data pre-processing step by enabling a ranking of transformations according to their relevance to the analysis. The ranking is made possible through the following three phases, shown in Figure 5.

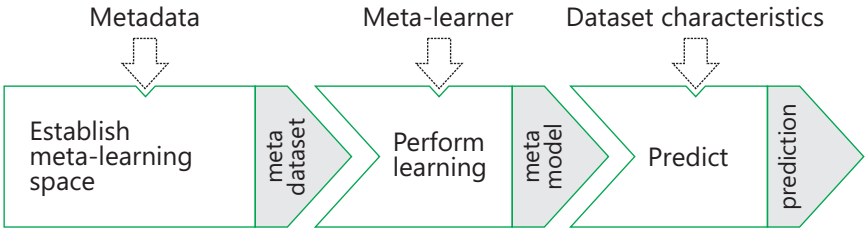

Figure 5: Meta-learning process

First, a meta-learning space is established using metadata [49]. The metadata consists of dataset characteristics along with some performance measures for data mining algorithms on those particular datasets. Then, the meta-learning phase generates a model (i.e., predictive meta-model), which defines the area of competence of the data mining algorithm [50]. Finally, when a transformed dataset (i.e., a transformation was applied on the dataset) arrives, the dataset characteristics are extracted and fed into the predictive meta-model, which predicts the performance of the algorithm on the transformed version of the dataset. At this point, we are able to obtain predictions for different transformed datasets (e.g., different transformations applied to the same dataset). Ordering the predictions from highest to lowest allows ranking transformations according to their predicted impact on the given dataset. This concludes the prediction phase.

In summary, to enable meta-learning, the first thing to do is to create a dataset for each classification algorithm, that is, a matrix-like structure consisting of variables/features/predictors and a response. The variables of this dataset are dataset characteristics — in this case, characteristics of the transformed datasets (e.g., number of instances, number of missing values, etc.). The response is a performance metric of the classification algorithm (e.g., predictive accuracy). Since all these are metadata, this dataset is called a **meta-dataset**. Consequently, its variables are referred to as

No	Name	Type	Modifiable
1..2	[Number|Percentage] of Continuous Attributes	Continuous	Yes
3..6	Min[Means|Std|Kurtosis|Skewness] of Cont. Att.	Continuous	Yes
7..10	Mean[Means|Std|Kurtosis|Skewness] of Cont. Att.	Continuous	Yes
11..14	Max[Means|Std|Kurtosis|Skewness] of Cont. Att.	Continuous	Yes
15..17	Quartile [1|2|3] of Means of Continuous Attributes	Continuous	Yes
18..20	Quartile [1|2|3] of Std of Continuous Attributes	Continuous	Yes
21..23	Quartile [1|2|3] of Kurtosis of Cont. Att.	Continuous	Yes
24..26	Quartile [1|2|3] of Skewness of Cont. Att.	Continuous	Yes
27	Number of Categorical Attributes	Categorical	Yes
28	Number of Binary Attributes	Categorical	Yes
29	Percentage of Categorical Attributes	Categorical	Yes
30	Percentage of Binary Attributes	Categorical	Yes
31..33	[Min|Mean|Max] Attribute Entropy	Categorical	Yes
34..36	Quartile [1|2|3] Attribute Entropy	Categorical	Yes
37..39	[Min|Mean|Max] Mutual Information	Categorical	Yes
40..42	Quartile [1|2|3] Mutual Information	Categorical	Yes
43	Equivalent Number of Attributes	Categorical	Yes
44	Noise to Signal Ratio	Categorical	Yes
45..48	[Min|Mean|Max|Std] Attribute Distinct Values	Categorical	Yes
49	Number of Instances	Generic	Yes
50	Number of Attributes	Generic	Yes
51	Dimensionality	Generic	Yes
52,53	[Number|Percentage] of Missing Values	Generic	Yes
54,55	[Number|Percentage] of Instances with Miss. Vals.	Generic	Yes
56	Number of Classes	Generic	No
57	Class Entropy	Generic	No
58,59	[Minority|Majority] Class Size	Generic	No
60,61	[Minority|Majority] Class Percentage	Generic	No

Table 4: Meta-features (Dataset characteristics)

meta-features and the response variable is named **meta-response**. Furthermore, the process of learning on top of this meta-dataset is referred to as **meta-learning** and the learning algorithm used, is referred to as **meta-learner**. The meta-features, the meta-response and the meta-learner are the key ingredients, and we will delve into details of each one of them, in the following sections.

4.1. Meta-features

Meta-features characterize a dataset, and two main classes have been proposed:

- *General measures:* include general information related to the dataset at hand. To a certain extent they are conceived to measure the complexity of the underlying problem. Some of them are: the number of instances, number of attributes, dataset dimensionality, ratio of missing values, etc.
- *Statistical and information-theoretic measures:* describe attribute statistics and class distributions of a dataset sample. They include different summary statistics per attribute like mean, standard deviation, class entropy, etc.

Additional meta-features measuring the association between the predictors and the response have been proposed. These measures are grouped into the *Landmarking and Model-based* class [51, 52]. This class is related to measures asserted with simple machine learning algorithms, so called *landmarkers*, and their derivatives based on the learned models. They include error rates and area under the roc curve (AUC) values obtained by landmarkers such as 1NN, DecisionStump or NaiveBayes. When performed on bigger datasets however, these simple machine learning algorithms introduce significant computational costs [42, 53]. Hence, we do not consider *Landmarking and Model-based* measures as dataset characteristics and they do not participate as meta-features in our experiments.

The meta-features we specifically consider are shown in Table 4. These are the set of meta-features extracted from OpenML[6] [9], which is an open science platform developed with the aim of allowing researchers to share their datasets, implementations, and experiments (machine learning and data mining) in a way that they can easily be found and reused by others. It is the biggest source of data and metadata for advancing meta-learning studies. Note that similar statistics have been used by other works too [54].

In Table 4, column *Type*, specifies the type of the meta-feature, and it can be `Continuous` — the meta-feature can be extracted only from datasets that contain attributes of continuous type, `Categorical` — the meta-feature can be extracted only from datasets that contain attributes of categorical type, `Generic` — the meta-feature can be extracted from any dataset, regardless of its attributes types.

Column *Modifiable* indicates whether the meta-features are modifiable through the transformations used (listed in Table 1). If meta-features are not modifiable, they are not considered, because they remain constant and they do not reflect the impact of transformations.

Note that the ultimate goal is to predict the impact of transformations, and the impact per se is measured as the relative change of the performance of the algorithm before and after the transformation is applied. To this end, to the set of meta-features we consider, we attach also the base performance of the classification algorithm (i.e., the performance before the transformation is applied) and in addition we add features that capture the difference between the meta-features before and after the transformation is applied. We call these features **delta meta-features**. As a result, every meta-feature has its corresponding delta meta-feature. For instance, let us say that in a given dataset, before applying a transformation, the *number of continuous attributes* is 5. Assume we apply a transformation that is discretizing only one continuous attribute, then, the number of continuous attributes becomes 4 and thus the delta of this feature is -1 (i.e., the *delta of the number of continuous attributes*).

Taking the deltas into account the total set of meta-features becomes large. We apply meta-feature extraction and selection in order to select only the most informative (with more predictive

[6] See `https://www.openml.org/search?type=measure` for more detailed explanations of the meta-features

Measure	Formula	
Accuracy	$\frac{TP+TN}{TP+FP+FN+TN}$	(1)
Precision	$TP/(TP+FP)$	(2)
Recall	$TP/(TP+FN)$	(3)
AUC	$Prob(X2 > X1)$	(4)

TN - True Negatives; TP - True Positives; FN - False Negatives; FP - False Positives; X1, X2 - Score functions of the classes

Table 5: Performance evaluation measures for classification algorithms

power) meta-features. Details on the meta-feature extraction/selection performed can be found in our previous work [53].

4.2. Meta-response

The goal of meta-learning is to correctly predict the impact of transformations on the performance of machine learning algorithms. Different measures can be used to evaluate the performance of machine learning algorithms. Since we are dealing with classification problems, and hence the algorithms we consider are of classification type, the performance is usually measured in terms of *predictive accuracy*, *precision*, *recall*, or AUC [55]. Moreover, classification algorithms are usually evaluated using either 10-fold cross-validation or leave-one-out validation (LOOV) [56]. In Table 5, formulas for calculating these measures are given. Briefly, *Accuracy* is a measure of the overall effectiveness of a classifier. *Precision* is the class agreement of the instance labels with the positive labels given by the classifier. *Recall* measures the effectiveness of a classifier to identify positive labels. Finally, one can think of AUC as the classifier's ability to avoid false classification. For more details regarding these measures and how they extend to multi-class classification problems, we refer the reader to [57].

These measures are collected before and after the transformations have been applied. The relative change between the performance obtained after the transformation and the base performance (performance obtained before the transformation) is the impact of a transformation on the predictive power of a classification algorithm, and this is the meta-response. Based on the meta-features and delta meta-features mentioned previously, the goal of the meta-learner is to correctly predict this impact, which can be *positive* — the performance increases after the transformation, *negative* — the performance decreases after the transformation, and *zero* — the performance remains the same.

4.3. Meta-learner

Given the meta-features (including delta meta-features) and the meta-response, the next step is to define the meta-learning problem. Since the meta-response — the impact of transformations, is of continuous (numeric) type, the meta-learning problem naturally fits to a regression problem. However, since we are mainly interested on the sign of impact and not on the exact amount of

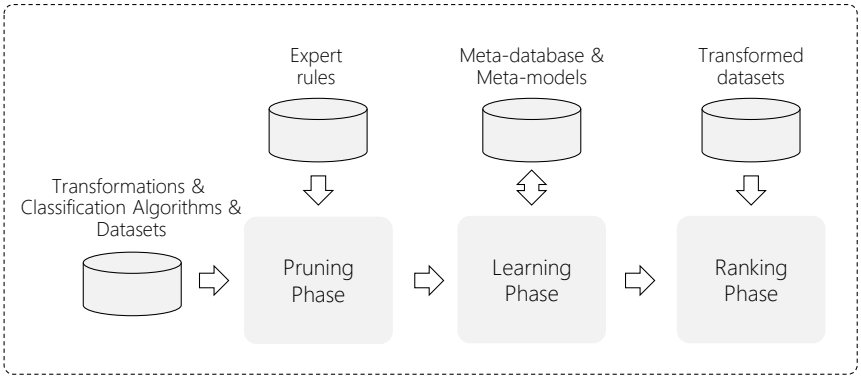

Figure 6: Overview of PRESISTANT

impact, the problem may as well be defined as a classification problem, where three classes would be required, *positive*, *negative*, and *zero*. In general, regardless of the type of the meta-learning chosen, the problem can be defined as follows . Given algorithm A and a limited number of training data $D = (\mathbf{x}_1, y_1)...(\mathbf{x}_n, y_n)$, the goal is to find a meta-learner with optimal/good generalization performance. Generalization performance is estimated by 10-fold cross-validation, which splits the training data into n (i.e., $n = 10$) partitions $D^{(1)}_{valid}, ..., D^{(n)}_{valid}$, and sets $D^{(i)}_{train} = D \backslash D^{(i)}_{valid}$ for $i = 1, ..., n$. Note that $\mathbf{x} \in x_1, x_2...x_n$ are the meta-features and delta meta-features and y_i is the impact of the transformation on the performance of algorithm A run on that particular transformed dataset. Hence, $\mathbf{x}$ and y altogether are the extracted metadata. Since y consists of 4 different performance measures (shown in Table 5) for algorithm runs, we build meta-datasets for each specific measure separately (we discuss only the results on predictive accuracy). Then, for each meta-dataset, we generate meta-models — using a meta-learner.

5. PRESISTANT

When dealing with a classification problem, the data analyst has to choose between a large number of classification algorithms, and a plethora of different pre-processing options. Typically, the aim is to combine the latter two, such that the obtained results allow for valid decision making. Yet, finding such a combination is challenging for non-expert users.

Therefore, in this regard, once the classification algorithm is chosen (i.e., one of the algorithms considered), our tool PRESISTANT (refer to [58] for a demonstration paper) assists the user by reducing the number of pre-processing options to only a set of relevant ones (i.e., operators that have positive impact). To do this, PRESISTANT uses a method consisting of three phases, shown in Figure 6.

In the first phase, rules are applied to prune the irrelevant transformations such that the search space is reduced. In the second phase, a model is trained to learn the impact of transformations on the performance of classification algorithms. Finally, in the third phase, the trained model is used to rank the newly arriving transformed versions of datasets.

5.1. Pruning phase

Given that there is an overwhelming number of different transformations that can be applied to a dataset, in Section 2.2.1, we argued that simple rules can help on discarding transformations that

Algorithm 1 Establish meta-database

Output: **meta_db**[1..#*algs*][1..#*trans*][1..#*metadata*]; ▷ meta-database; meta-dataset per classification alg.

```
1:  function CreateMetaDB(datasets,transformations,classificationAlgs)
2:      metadata = []; ▷ the set of features to be collected
3:      ds_mf = []; ▷ meta-features of the non-transformed dataset
4:      trans_ds_mf = []; ▷ meta-features of the transformed dataset
5:      Δmf = []; ▷ delta meta-features
6:      for each algorithm alg in classificationAlgs do
7:          for each dataset ds in datasets do
8:              ds_mf = ComputeMetaFeatures(ds); ▷ see Table 4
9:              ds_pm = GetPerformanceWith10FoldCV(alg,ds); ▷ see Table 5
10:             for each transformation tr in transformations do
11:                 trans_ds = ApplyTransformation(tr,ds);
12:                 trans_ds_mf = ComputeMetaFeatures(trans_ds);
13:                 Δmf = trans_ds_mf − ds_mf;
14:                 trans_ds_pm = GetPerformanceWith10FoldCV(alg,trans_ds);
15:                 mr = |trans_ds_pm − ds_pm| / ds_pm; ▷ relative diff., meta-response
16:                 metadata = trans_ds_mf ∪ Δmf ∪ ds_pm ∪ mr;
17:                 meta_db[alg][trans_ds] = metadata;
18:     return meta_db;
```

have no impact. This translates to having a repository of Expert Rules (cf. Figure 6), that can be extended to contain any types of rules (e.g., the types of rules used in IBM SPSS Modeler[7]), that may help on reducing the number of potential transformations to be applied on datasets. Our first basic set of rules are derived from the experiments whose results were shown in Figure 3, where for instance we define rules in order to exclude Standardization and Normalization when considering algorithms like, IBk, Logistic, J48, and PART.

5.2. Learning phase

Two important activities are performed in the learning phase. First, a meta-database (i.e., set of meta-datasets) is generated for all the classification algorithms considered (cf. Algorithm 1), and then on top of it, a learning algorithm is applied (cf. Algorithm 2). As a result, a statistical model (meta-model) is generated for every classification algorithm considered.

The inputs required to construct the meta-database are datasets, transformations — that are likely to improve the performance of classification algorithms, and the classification algorithms in consideration.

For the sake of simplicity, let us consider that we want to create the meta-dataset for a single classification algorithm. In line 8 of Algorithm 1, we first extract the dataset characteristics (i.e., meta-features from the original non-transformed datasets). Next, we apply all the available transformations to all the datasets and hence obtain transformed datasets, see line 11. We extract

[7]Rule based method for automated data preparation used by IBM: `https://www.ibm.com/support/knowledgecenter/en/SS3RA7_15.0.0/com.ibm.spss.modeler.help/idh_idd_adp_objective.htm`

Algorithm 2 Create meta-models

Input: datasets ▷ available datasets of classification type
transformations ▷ available transformations to be applied
classificationAlgs ▷ available classification algorithms
Output: models ▷ meta-model for each algorithm

```
1: function PerformMetaLearning
2:     meta_db = CreateMetaDB(datasets,transformations,classificationAlgs);
3:     meta_learner = RandomForest();                    ▷ a meta learner of choice
4:     for each algorithm alg in classificationAlgs do
5:         models[alg] = ApplyMetaLearner(meta_learner,meta_db[alg]);
6:     return models;
```

the meta-features from the transformed datasets in line 12, and take the difference between them and the meta-features from the original non-transformed datasets in line 13. Like this, we obtain the delta meta-features. Furthermore, to both original non-transformed datasets — line 9, and the transformed ones — line 14, we apply the classification algorithm and then take the relative difference between their corresponding performance measures (e.g., predictive accuracy) — line 15. The latter is the meta-response, which together with the meta-features of the non-transformed version of the dataset, the delta meta-features, and the performance measure of the original dataset compile the complete set of metadata (list of features that will be used in the learning phase) — see line 16.

Once a meta-dataset is obtained for each classification algorithm, next a learning algorithm (i.e., meta-learner) is applied — line 6 of Algorithm 2. As a result, a meta-model (i.e., statistical model) for each of the classification algorithms is obtained. PRESISTANT uses the Random Forest [59] algorithm as meta-learner. The XGBoost [60] algorithm was also tested as meta-learner. Even though similar results were obtained, we opt for the models obtained from Random Forests because they are easier to interpret.

Lastly, note that since separate models are built for each algorithm, the predictive power of meta-features slightly differs depending on the algorithm [53]. We refer the reader to our previous work [53] for a detailed analysis of the most important meta-features and their predictive power.

5.3. Ranking/Recommending phase

The recommending phase starts when a user wants to analyze a dataset. She selects an algorithm to be used for the analysis and the system automatically recommends transformations to be applied, such that the final result is improved. This phase is described in Algorithm 3. In Algorithm 3, first the meta-features and the performance of the classification algorithm are extracted from the original non-transformed dataset in lines 3 and 4, respectively. Next, different transformations are applied to the dataset and from each transformed version of the dataset the necessary features (i.e., meta-features, delta meta-features) are computed — see lines 5-9. These features are then fed to the predictor in line 10. The predictor in line 10, applies the meta-model to the extracted features in order to find the predicted impact of a transformation on the performance of the algorithm.

After the predicted impacts are obtained for all the transformations, they are ranked in descending order using the probabilities of being positive, which are provided by the model, in line 11.

Algorithm 3 Recommend transformations

Input:
models; ▷ meta-model for each algorithm
transformations; ▷ available transformations to be applied
ds ▷ new dataset chosen by the user
Output: transformations; ▷ trans. ordered according to predicted impact

1: **function** RANKTRANSFORMATIONS(**datasets**,**transformations**,**classificationAlgs**)
2: **predictions** = [][]; ▷ predictions for transformed datasets
3: **ds_mf** =COMPUTEMETAFEATURES(*ds*);
4: *ds_pm* =GETPERFORMANCEWITH10FOLDCV(*alg*,*ds*);
5: **for** each transformation *tr* in *transformations* **do**
6: *trans_ds* = APPLYTRANSFORMATION (*tr*,*ds*);
7: **trans_ds_mf** =COMPUTEMETAFEATURES(*trans_ds*);
8: **Δmf** = **trans_ds_mf** − **ds_mf**;
9: **features** = **trans_ds_mf** ∪ **Δmf** ∪ *ds_pm*;
10: **predictions**[*tr*] = APPLYMODEL(**features**,**models**[*classAlg*]); ▷ predict perf.
11: **transformations** =RANKBYPROBABILITIES(**predictions**,*desc* = *true*);
12: **return transformations**;

To this end, a strong feature of PRESISTANT is that it actually only predicts the performance of the algorithms, otherwise the classification algorithms need to be applied for real on the transformed datasets. The former, as shown in Table 6, is orders of magnitudes faster than the latter.

Furthermore, the number of total transformations executed per dataset determines the level of interactivity that one can get with PRESISTANT. The cost of executing the transformations, as shown in Table 7, is very cheap, unless more complex transformations like PCA are considered. However, as we already mentioned in Section 5.1, the task of an expert in the pruning phase would be to define rules that would filter the useless transformations and at the same time in a sense configure the overall level of interactivity.

6. Evaluation

We perform experimental studies to evaluate the performance of our tool. In particular, the aim of the experiments are three-fold:

Algorithm	Alg. Execution	Predictions
weka.J48	4658	0.1
weka.NaiveBayes	1530	0.1
weka.PART	14144	0.1
weka.Logistic	28880	0.1
weka.IBk	7624	0.1

Table 6: Comparison of average run times (ms) per dataset, real executions vs predictions

Transformation	Exec. Time
Discretization	14.65
NominalToBinary	4.63
Normalization	0.50
Standardization	0.63
PCA	980

Table 7: Average run time (ms) of transformations per dataset

Real \ Predicted	Positive	Negative	Zero
Positive	TP	FN_P	$F0_P$
Negative	FP_N	TN	$F0_N$
Zero	FP_0	FN_0	T0

a)

Real \ Predicted	Positive	Negative	Zero
Positive	TP	FNP	
Negative	FP′	TNP	
Zero			

b)

Table 8: Confusion matrices

1. Asses the performance of PRESISTANT in terms of the quality of the predictions from the meta-learner perspective. We try to answer the question "How good are the predictions?" (cf. Section 5.1).

2. Assess the gain obtained by recommendations, from the user perspective. We try to answer the question "How valuable are the recommendations?" (cf. Section 5.2).

3. Assess the performance of the recommendations of PRESISTANT, compared to the transformations picked by humans in a set of randomly selected classification problems. We try to answer the question "How difficult is to find the correct transformation in practice?" (cf. Section 5.3).

To enable the use of the entire set of datasets in the experiments, we use the 10-fold cross-validation method. This entails that for each classification algorithm considered, when building the meta-models, if a dataset is used in testing the same is not considered in the training. Furthermore, in this work we discuss the results obtained when treating the meta-learning problem as a classification task. We refer the reader to our previous work [48], in which the meta-learning problem is treated as regression.

6.1. Evaluation of the quality of predictions

Predictions provided by the meta-model enable the ranking of transformations. The list of recommended transformations can be very large in case a lot of transformations are considered. One may be interested in the whole set of transformations (e.g., "recommending all good items" in collaborative filtering), or only on the top-K transformations, K being an arbitrary number (e.g., "recommending some good items" in collaborative filtering). The latter is more realistic since the greater the ranked position, the less valuable a transformation is for the user, because the less likely it is that the user will examine the transformation due to time, effort and cumulated information from transformations already seen/applied [61]. We performed evaluations both considering the whole set of transformations and considering only the top-K.

6.1.1. Evaluation of the quality of the whole set of transformations

When treated as classification, the meta-learning problem translates to a multi-class classification problem with three classes (i.e., positive, negative, zero) in the response variable. Given that it is a multi-class problem and knowing that all the classes do not have the same importance, we cannot use the traditional binomial confusion matrix for the evaluation. For instance, regarding

the importance of classes, miss-predicting a zero transformation does not have the same impact as miss-predicting a positive or negative transformation.

Therefore, in Table 8a), we devise a confusion matrix that consists of two parts. The inner (green) part is the traditional confusion matrix for the positive and negative predictions and the outer (orange) part, is the one that takes into account the zero class.

Furthermore, since datasets have varying numbers of attributes, they do not have the same number of transformations applied to them i.e., some datasets can have more transformations than others (cf. Table 1). To give equal importance to every dataset, regardless of the number of transformations, we assign weights to their corresponding transformations (i.e., transformed versions of the same dataset): $w_{T_d} = 1/|T_d|$, where $|T_d|$ is the total number of transformations applied to dataset d. Like this, each dataset has weight equal to 1.

Using the matrix shown in Table 8a), per dataset d and for the set of its transformations T_d with their corresponding weights w_{T_d}, we evaluate our system calculating the Predictive accuracy (PA_d), Precision (Pr_d), Overall recall (OR_d) and F-modified ($F\text{-}m_d$), defined as follows:

$$PA_d = \frac{TP + TN}{TP + FN_P + FP_N + TN} \quad Pr_d = \left(\frac{TP}{TP + FP_N} + \frac{TN}{TN + FN_P}\right)/2$$

$$OR_d = \frac{TP + FN_P + FP_N + TN}{TP + FN_P + FP_N + TN + (F0_P + F0_N)} \quad F\text{-}m_d = 2\left(\frac{PA_d \times OR_d}{PA_d + OR_d}\right)$$

where TP represents the number of true positives, FN_P the number of false negatives, FP_N the number of false positives, and TN the number of true negatives. Furthermore, $F0_P$ represents the number of transformations that are predicted as zero, but in reality they are positive, $F0_N$ represents the zero predicted transformations that are in reality negative. Finally, FP_0 are positive predictions that in reality have zero impact, FN_0 negative predictions that in reality have zero impact, and $T0$ are the true zeros. Notice, that FP_0 and FN_0 are less harmful than $F0_P$ and $F0_N$, since predicting a transformation as zero and then having a positive impact in real ($F0_P$), is worse than predicting a transformation as positive and then having zero impact in real (FP_0). The same applies for FN_0 when compared to $F0_N$.

The aforementioned measures are calculated for individual datasets d. Averaging the individual measures over all datasets with at least one relevant transformation, we obtain the mean Predictive accuracy PA, Precision Pr, Overall recall OR, and F-modified $F\text{-}m$. In the experiments performed on 533 datasets we obtained the results shown in Figure 7.

The results show that, on average, if a user selects any transformation from the whole list of possible transformations (T_d), the system provides an accuracy of 61%.

6.1.2. Evaluation of the quality of the top-K recommendations

Since real users are usually concerned only with the top part of the recommendation list, a more practical approach is to consider the number of a datasets' relevant transformations ranked in the top-K positions.

Many details need to be considered in order to perform a proper evaluation of the K positions.

First of all, for the sake of simplicity, let us use the confusion matrix shown in Table 8b), where we denote as True Non Positive ($TNP = TN + F0_N + FN_0 + T0$), a transformation that is predicted

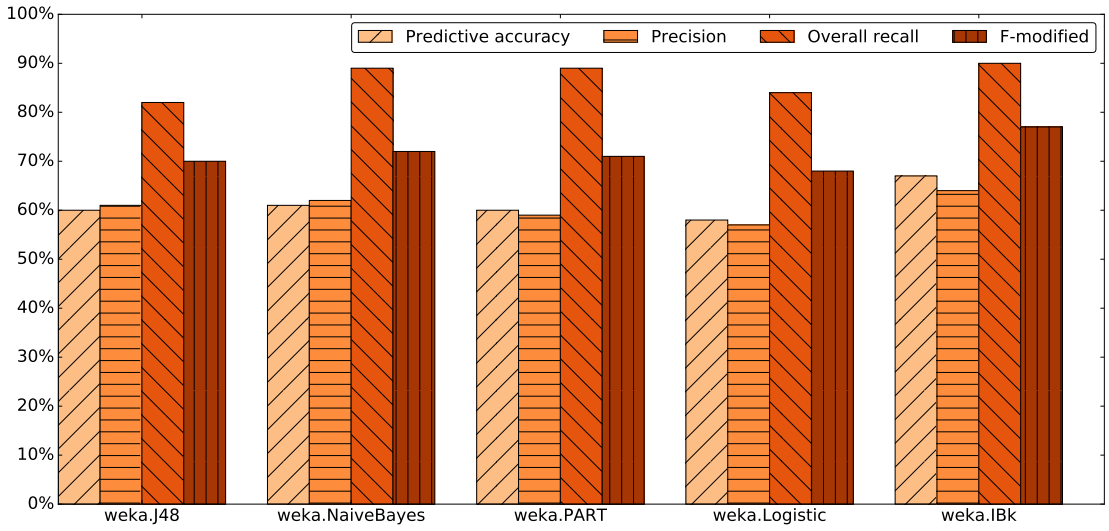

Figure 7: Results obtained when evaluating the whole set of transformations

as non-positive and it is non-positive in real (i.e, after executing the classification algorithm the transformation has no positive impact). False Non Positive ($FNP = FN_P + F0_P$), a transformation that is predicted as non-positive but is positive in real. Finally, FP', a transformation that is predicted as positive but in reality can have either negative or zero impact.

Next, notice that for each dataset, there can be L transformations that have real positive impact, and the system (in practice) recommends y transformations that are predicted to have a positive impact.

To be able to perform evaluations for all the datasets, including the datasets with L=0 (i.e., datasets that do not have any transformations that have real positive impact), and to be able to calculate average measures for datasets with different L, for all the positions in the ranking, we rank the transformations as follows: first we rank the y positively predicted transformations by their probability of being positive (the highest goes first). Next, we append the remaining real positive transformations (if any are left) up to L. Finally, we append all the remaining transformations ranked by their probability of being positive.

This ranking allows us to perform evaluations for each position K for any dataset with L real positive transformations. The results of the evaluations form a matrix of size $[L, K]$, and the possible evaluation we can have is shown in Table 9, where two possible scenarios are considered,

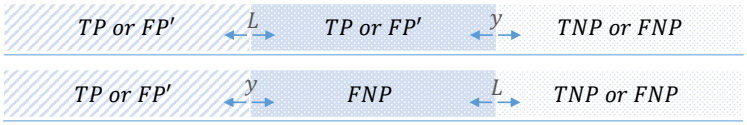

Table 9: Evaluation of our approach based on the chosen ranking (ordering of transformations)

1. **if** $y > L$ (i.e., we predict too many positive transformations), the current transformation in position c can be below y, where we can find either TP or FP' and above y, where we can find either TNP or FNP.
2. **if** $y \leq L$ (i.e., we predict too few positive transformations), the current transformation in

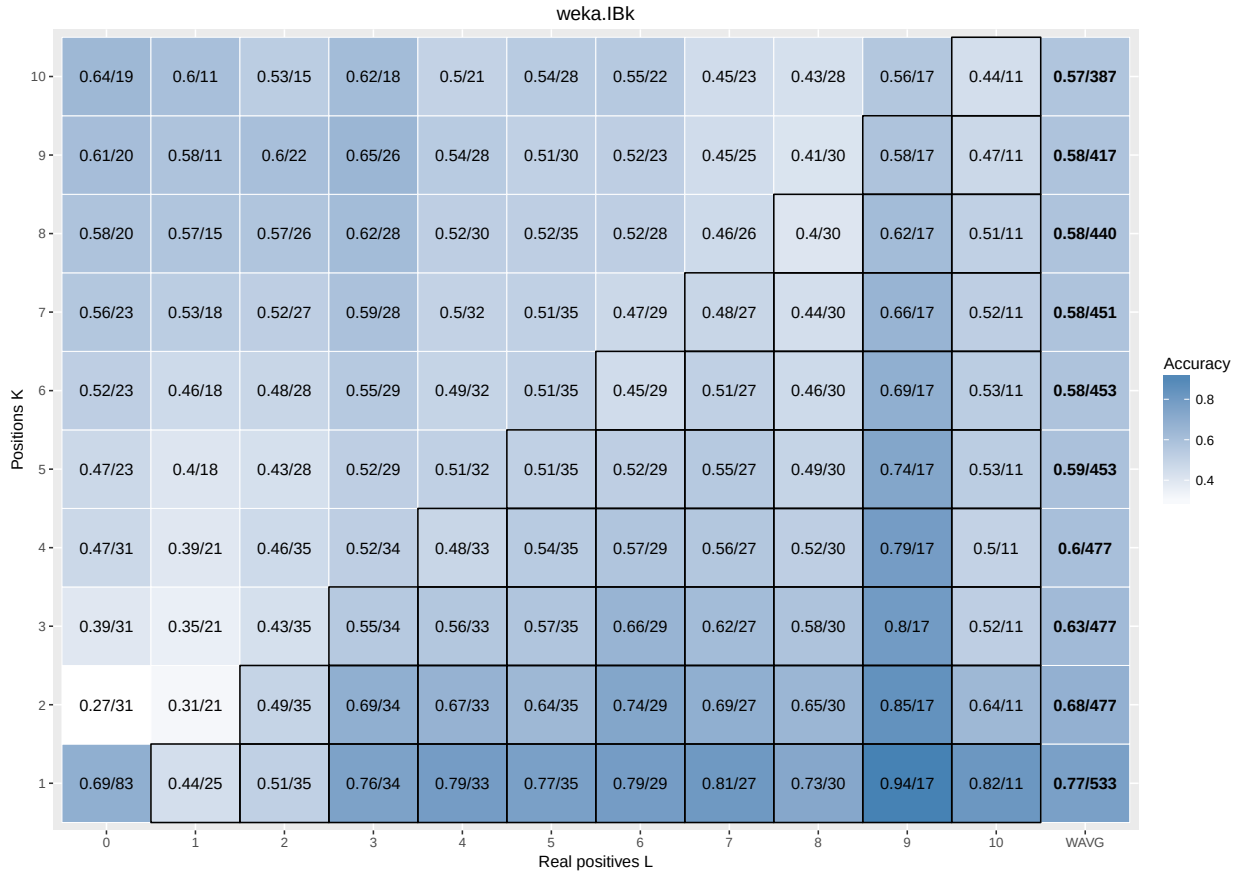

Table 10: Accuracy values obtained for the IBk classifier, for transformations in positions K in datasets with L real positive transformations. The numbers shown inside the cells, denoted as x/z, are the cumulative accuracies x, and the number of datasets z, which have at least K transformations and exactly L real positives. The last column, shows the values obtained after computing the weighted average for each position K for all possible L

position c can be: below y, between y and L, and above L. Below y, we can find either TP or FP'. Above L, we can find either TNP or FNP. Finally, between y and L we can only find FNP, that is $TP = 0$, since here we have the transformations that are predicted as non-positive ($c > y$), but in reality they have positive impact,

Using the aforementioned, we can calculate accuracy measures, where below the diagonal we can compute the ratio of true positives ($TP/(TP + FP' + TNP + FNP)$), and above the diagonal we can calculate the ratio of true non positives ($TNP/(TP + FP' + TNP + FNP)$). Note that because of the ordering enforced (as shown in Table 9), we can have TP only below the diagonal and TNP only above the diagonal.

In Table 10, we show the results obtained for algorithm IBk. The numbers shown inside each cell $[L, K]$, denoted as x/z, are the cumulative accuracies x, and the number of datasets z (with at least K transformations and exactly L real positives). For instance, in the cell [L=1,K=1], we have an accuracy of 44%. That is for datasets with only one real positive transformation (there are 25 such datasets), in 44% of the cases we correctly rank in position K=1 the transformations that have positive impact. This implies that it is rather difficult to find a positive transformation on datasets that have only 1 real positive transformation. However, for instance, for L=9 in position

K=1, the accuracy is much higher (i.e., 94%). Therefore, since the cells are colored based on the accuracy (the darker the color the higher the accuracy), below the diagonal they become darker as L grows. This means that PRESISTANT obviously performs better when the number of real positives is higher. Furthermore, the cells in the bottom part are darker than the rest, so the results are better in the first (top) K positions.

The last column of Table 10 shows the weighted average values of the accuracies obtained for each position K. The accuracies are weighted by the number of datasets in each L; the results shown are computed for all L.

Finally, as a summary, in Table 11 we show the weighted average results of the accuracies for all L, in position K=1, for all the algorithms considered[8].

Algorithm	**WAVG in K=1**	**#Datasets**
J48	0.62	533
Naive Bayes	0.78	533
PART	0.67	533
Logistic	0.57	480
IBk	0.77	533

Table 11: Weighted average values for all L in K=1

Comparison with a random pick

To evaluate the performance of our approach, we compare it to the approach of a user randomly choosing a transformation to apply. For the latter, given the data, we need to find the probability of having TP below the diagonal and having TNP above the diagonal.

Finding the probability of having TP below the diagonal in the K^{th} position, translates to the problem of finding the probability of picking a positive transformation in K draws, from a bag consisting of positive, negative, and zero (neutral) transformations. This follows a hyper-geometric distribution, and the expected value of TP in a cell $[L, K]$ below the diagonal is calculated as $\mu_{TP} = K\frac{L}{|T_d|}$, where $|T_d|$ denotes the total number of transformations in dataset d. The expected value of TNP is calculated as $\mu_{TNP} = K\frac{L-|T_d|}{|T_d|}$.

To make a fair comparison with our approach, we need to assume the same ordering. Hence, the values to be calculated are shown in Table 12, where y' is the expected number of positive predictions a dataset can have, which is calculated as the ratio of the transformations of dataset d, given the proportion of all real positives we can have for algorithm a, $y' = |T_d|P_a(Positive)$. Note that the probabilities of being positive for each algorithm a, i.e., $P_a(Positive)$, are found using the distributions of the impacts in Figure 2.

In Table 12, again two scenarios are considered:

1. **if** $y' > L$ (i.e., the random picks too many positive transformations), below y' we take the probability of being TP, and above y' we take the probability of being TNP.
2. **if** $y' \leq L$ (i.e, the random picks too few positive transformations), for the current transformation c below y', we calculate the probability of being TP. For the transformation positioned between y' and L, the probability of being TP is 0. Finally, for the transformation above L, we calculate the expected TNP based on the expected positive predictions y',

[8]For the complete results of all the algorithms, visit: `http://www.essi.upc.edu/~bbilalli/presistant.html`

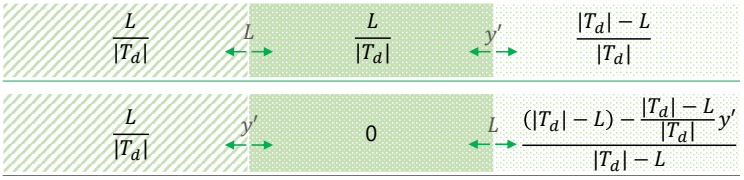

Table 12: Evaluation of the random pick based on the chosen ranking

More precisely, given the conditions in Table 12, the probabilities for each cell $[L, K]$, are calculated using the following function,

$$P(L,K) = \begin{cases} (min(K,y')\frac{L}{|T_d|} + max(0,K-y')\frac{|T_d|-L}{|T_d|})/K, & \text{if } y' > L \\ (min(K,y')\frac{L}{|T_d|} + max(0,K-L)\frac{(|T_d|-L)-(\frac{|T_d|-L}{|T_d|}y')}{|T_d|-L})/K, & \text{if } y' \leq L \end{cases}$$

To show whether the values obtained by our approach are significant, we performed a binomial distribution test comparing the true positives obtained by our approach with the total number of datasets with respect to the theoretical probabilities obtained by the random pick. The results obtained are shown in Table 13, where the color of the cell denotes whether the value obtained is significant or not (white means significant). We consider the value to be significant if it is $p <= 0.01$.

It can be observed that significant values are obtained for most of the cells below the diagonal where the accuracy with regards to TP is measured. Furthermore, it is worth to mention that:

- observing the bottom left-most cell, it can be noted that the system increases the chance of finding the transformations that do not positively impact the analysis (the system may suggest avoiding those transformations)
- the bottom right-most cell indicates that for position $K = 1$, we almost double the accuracy compared to the random pick (77% versus 41%). Moreover, the accuracy obtained for the whole set of transformations for IBk was 67% and for top-1, becomes 77%
- the probabilities of the random pick start to become higher above the diagonal, due to the fact that it is easier to guess negative transformations as you go down in the ranking
- significance values are also impacted by the sample sizes (number of datasets), which are different for each L and they may also vary for different Ks. Yet, observe the last column where the weighted averages are shown. The calculations are done for all the datasets on each K, and the values obtained are significant.

6.2. Evaluation of the gain obtained from recommendations of PRESISTANT

In the Information Retrieval domain, different measures that calculate the gain obtained from a ranked result have been proposed. The most popular one among them is the Discounted Cumulative Gain (DCG) [61]. The assumption is that the greater the ranked position, the less valuable the item (i.e., transformations) is for the user, because it is less likely that the user will examine it. Thus, DCG uses a discounting function that progressively reduces the gain as the rank increases. To compute DCG, a permutation/ordering π of the gain values G_{T_d} (i.e., the real gain in terms of accuracy induced by the transformation) on the entire list of transformations in a given dataset d, results in the ordered list of gains G_{π,T_d} (a vector of length N, where N is $|T_d|$). In particular, we are interested in the permutation that sorts according to our predicted scores (i.e., predicted

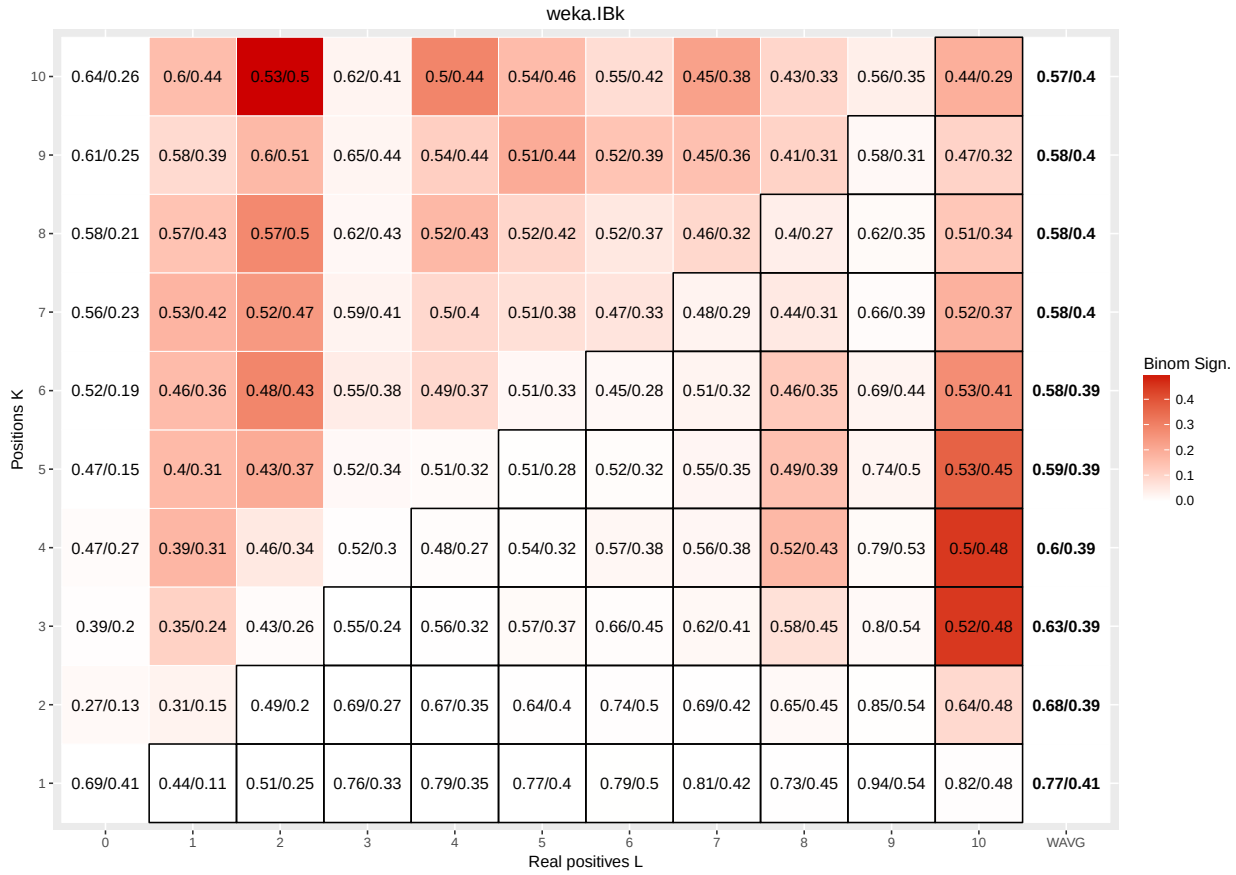

Table 13: Significance values obtained when comparing results obtained with meta-learning and the random pick, for the IBk classifier. The numbers shown inside the cells, denoted as x/v, are the cumulative accuracies x obtained using our approach, and the random pick probabilities v, for datasets with at least K transformations and exactly L real positives. The last column shows the averages for our approach compared to the random pick, in each position K, weighted by the number of datasets in each L

probabilities) for the various transformations in a dataset: G_{rec,T_d} denotes the recommended list or gains in descending order of the predicted scores of the transformations (highest ranked transformation is first in list). Moreover, the best (G_{best,T_d}) and worst (G_{worst,T_d}) possible permutations are also of interest.

For the general case DCG is computed as [62]:

$$DCG_{G_{\pi,T_d}} = \sum_{i=1}^{N} \frac{G_{\pi,T_d}[i]}{log_2(i+1)}$$

The calculations are performed for each dataset and we obtain values for the recommended ranking ($DCG_{G_{rec,T_d}}$), the best ranking ($DCG_{G_{best,T_d}}$), and the worst ranking ($DCG_{G_{worst,T_d}}$).

To obtain a relative value, we normalize the gain obtained by our recommendations in the following way:

$$nDCG_d = \frac{DCG_{G_{rec,T_d}} - DCG_{G_{worst,T_d}}}{DCG_{G_{best,T_d}} - DCG_{G_{worst,T_d}}},$$

Algorithm	$\overline{nDCG}$		#Datasets[a]
	All trans.	Top-1	
J48	0.72	0.78	475
Naive Bayes	0.78	0.85	476
PART	0.73	0.79	476
Logistic	0.63	0.66	421
IBk	0.77	0.85	475

[a]Number of datasets with at least 1 relevant (non-neutral) transformation

Table 14: Normalized discounted cumulative gain values

This normalized value can be interpreted as a percentage of how close we are to the best ranking, and it is calculated for each dataset. Averaging the individual measures over all datasets with at least one relevant (non-neutral) transformation we obtain the mean $\overline{nDCG}$. This measure can again be calculated for the whole set of transformations or for the top-K. In Table 14, we provide the results obtained for the whole set of transformations and for top-1, for all the classification algorithms considered.

6.3. Evaluation of the performance of PRESISTANT compared to humans

In this section, we discuss the results obtained in an empirical evaluation with humans. The importance of evaluations with real users has already been acknowledged in previous works [63].

Our experiment, in the form of a quiz[9], is defined as follows: given a dataset, a classification algorithm, and a set of applicable transformations over the dataset, find the transformation that has more positive impact on the classification accuracy of the algorithm. If none of the transformations is ought to have positive impact, pick option "None".

For the answer to be correct, users must find only one positive transformation among the possibly many positive transformations per dataset. Whereas, PRESISTANT's top recommended transformation must be among the positive ones. In both cases, score 1 is assigned to the correct, and score 0 to the incorrect answer.

We performed experiments with 4 randomly selected datasets[10] (cf. Table 15), and the 5 classification algorithms that PRESISTANT supports. The maximum number of participants per dataset was 39, and their background varied between users with "No knowledge" in data mining — 15.38%, "Basic knowledge" — 43.59%, "Intermediate knowledge" — 38.47%, and "Expert knowledge" — 2.56%. Most of the participants held master's degrees in fields related to Computer Science — 69.24%. Others held PhD's in some Computer Science field — 15.38%, and the rest were undergraduate students in Computer Science — 15.38%. More than half of the participants were not students and they are currently pursuing their professional careers in either companies or academia, always in the field of data management and analytics.

The average scores obtained per algorithm by real users and PRESISTANT, are compared in Table 16. PRESISTANT scored on average 2.5 times better than humans, showing its effectiveness in real scenarios. More interestingly, however, users performed worse than they were expected

[9]`http://www.essi.upc.edu/~bbilalli/presistant.html#quiz`

[10]Datasets are retrieved from the UCI repository: `https://archive.ics.uci.edu/ml/datasets.html`

Dataset	#Participants
autos	31
ecoli	37
diabetes	30
flags	39

Table 15: UCI datasets used in the experiments

Algorithm	Users	PRESISTANT
J48	0.10	0.27
Naive Bayes	0.52	1.0
PART	0.21	0.51
Logistic	0.10	0.50
IBk	0.17	0.55
Total	**0.22**	**0.57**

Table 16: Average scores of users and PRESISTANT

(i.e, they score 0.22 and the expected score with a random pick is close to 0.3). We suspect this occurred because of the fact that users were biased towards selecting transformations that are of type Global — applied globally to all the compatible attributes of a dataset. Indeed, 7 out of top 10 most frequently picked transformations were either of type Global or "None" — 57.08%. Yet, out of the transformations shown to the user on average, only 37% of them were of type Global and the rest were Local.

Another interesting observation is that when results are broken down and PRESISTANT is only compared to the Expert category, PRESISTANT is still 1.5 times better on average for all the algorithms.

Thus overall, the results indicate that in practice it is difficult to find the transformations that positively impact the analysis and that there is obvious need for user support.

7. Conclusions and future work

In this work, we addressed the problem of assisting non-expert users to perform pre-processing with the goal of improving the final results of their classification tasks.

To provide assistance, we trained a model that learned the relationship between pre-processing operators and the performance of classification algorithms. To this end, we were able to rank transformations according to their impact on the final result of the analysis (i.e., the impact of transformations on the predictive accuracy of a classification algorithm). An extensive evaluation on hundreds of datasets and a set of classification algorithms, showed that our approach gives promising results. More specifically, we were able to observe that:

- even if a user randomly picks a transformation from the entire list of transformations ranked by PRESISTANT we obtain an average accuracy of 61%, for all the algorithms considered,
- recommending only the top-1 transformation, increased the accuracy to 68%,
- measuring the gain obtained from our ranking for all transformations using DCG, we were as close as 73% on average to the gain obtained from the best possible ranking (for all the algorithms considered),
- measuring the gain from the top-1 recommendations using DCG, we were as close as 79% on average to the gain obtained from the best possible ranking,
- in a set of randomly selected classification problems, PRESISTANT performed 2.5 times better than humans (mostly non-experts).

Finally, the results indicate that our tool PRESISTANT, can assist users to more effectively identify the pre-processing operators appropriate to their applications, and to achieve improved results.

As a future work, we see potential value on extending the list of the pre-processing operators and classification algorithms we have considered so far and we also plan to incorporate regression algorithms.

Acknowledgments. This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence - Doctoral College" (IT4BI-DC).

References

[1] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, From data mining to knowledge discovery in databases, AI Magazine 17 (3) (1996) 1–34.

[2] D. Michie, D. J. Spiegelhalter, C. C. Taylor, J. Campbell (Eds.), Machine Learning: Neural and Statistical Classification, Ellis Horwood, 1994.

[3] S. B. Kotsiantis, et al., Data Preprocessing for Supervised Learning, International Journal of Computer Science 1 (2006) 111–117.

[4] M. A. Munson, A study on the importance of and time spent on different modeling steps, ACM SIGKDD Exploration Newsletter 13 (2) (2012) 65–71.

[5] T. Furche, G. Gottlob, L. Libkin, G. Orsi, N. W. Paton, Data wrangling for big data: Challenges and opportunities, in: Proceedings of the International Conference on Extending Database Technology, EDBT '16, 2016, pp. 473–478.

[6] M. Lenzerini, Data integration: A theoretical perspective, in: Proceedings of the International Symposium on Principles of Database Systems, PODS '02, 2002, pp. 233–246.

[7] X. Chu, I. F. Ilyas, S. Krishnan, J. Wang, Data cleaning: Overview and emerging challenges, in: Proceedings of the International Conference on Management of Data, SIGMOD '16, 2016, pp. 2201–2206.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, et al., The weka data mining software: An update, ACM SIGKDD Explorations Newsletter 11 (1) (2009) 10–18.

[9] J. Vanschoren, J. N. van Rijn, B. Bischl, L. Torgo, OpenML: Networked science in machine learning, ACM SIGKDD Explorations Newsletter 15 (2) (2014) 49–60.

[10] J. W. Lee and, C. Giraud-Carrier, On the dangers of default implementations: The case of radial basis function networks, Intelligent Data Analysis 18 (2) (2014) 261–279.

[11] X. Chu, I. F. Ilyas, S. Krishnan, J. Wang, Data cleaning: Overview and emerging challenges, in: Proceedings of the International Conference on Management of Data, SIGMOD '16, 2016, pp. 2201–2206.

[12] T. Dasu, T. Johnson, Exploratory Data Mining and Data Cleaning, 1st Edition, John Wiley & Sons, Inc., New York, NY, USA, 2003.

[13] S. Kandel, A. Paepcke, J. Hellerstein, J. Heer, Wrangler: Interactive visual specification of data transformation scripts, in: Proceedings of the International Conference on Human Factors in Computing Systems, CHI '11, 2011, pp. 3363–3372.

[14] S. Kandel, A. Paepcke, J. Hellerstein, J. Heer, Wrangler: Interactive visual specification of data transformation scripts, in: Proceedings of the ACM Conference on Human Factors in Computing Systems, CHI '11, 2011, pp. 3363–3372.

[15] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, I. F. Ilyas, Guided data repair, PVLDB 4 (5) (2011) 279–289. doi:10.14778/1952376.1952378.
URL `http://portal.acm.org/citation.cfm?id=1952378\&CFID=12591584\&CFTOKEN=15173685`

[16] R. Singh, S. Gulwani, Learning semantic string transformations from examples, Proc. VLDB Endow. 5 (8) (2012) 740–751. doi:10.14778/2212351.2212356.
URL `http://dx.doi.org/10.14778/2212351.2212356`

[17] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, S. Xu, Data curation at scale: The data tamer system, in: Online Proceedings Biennial Conference on Innovative Data Systems Research, CIDR '17, 2013.

[18] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, N. Tang, Nadeef: A commodity data cleaning system, in: Proceedings of the International Conference on Management of Data, SIGMOD '13, 2013, pp. 541–552.

[19] F. Geerts, G. Mecca, P. Papotti, D. Santoro, The llunatic data-cleaning framework, Proceedings of the VLDB Endowment 6 (9) (2013) 625–636.

[20] M. Volkovs, F. Chiang, J. Szlichta, R. J. Miller, Continuous data cleaning, in: Proceedings of the International Conference on Data Engineering, 2014, ICDE '14, 2014, pp. 244–255.
[21] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, S. Yin, Bigdansing: A system for big data cleansing, in: Proceedings of the International Conference on Management of Data, SIGMOD '15, 2015, pp. 1215–1230.
[22] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, Y. Ye, Katara: A data cleaning system powered by knowledge bases and crowdsourcing, in: Proceedings of the International Conference on Management of Data, SIGMOD '15, 2015, pp. 1247–1261.
[23] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, K. Goldberg, Activeclean: Interactive data cleaning for statistical modeling, PVLDB 9 (12) (2016) 948–959.
[24] J. Morcos, Z. Abedjan, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, Dataxformer: An interactive data transformation tool, in: Proceedings of the International Conference on Management of Data, SIGMOD '15, 2015, pp. 883–888.
[25] Z. Jin, M. R. Anderson, M. Cafarella, H. V. Jagadish, Foofah: A programming-by-example system for synthesizing data transformation programs, in: Proceedings of the International Conference on Management of Data, SIGMOD '17, 2017, pp. 1607–1610.
[26] L. Berti-Equille, Learn2clean: Optimizing the sequence of tasks for web data preparation, in: The World Wide Web Conference, WWW '19, ACM, New York, NY, USA, 2019, pp. 2580–2586.
[27] A. Quemy, Data pipeline selection and optimization, in: Proceedings of the International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, DOLAP '19, 2019.
URL `http://ceur-ws.org/Vol-2324/Paper19-AQuemy.pdf`
[28] F. Serban, J. Vanschoren, J. Kietz, A. Bernstein, A survey of intelligent assistants for data analysis, ACM Computing Surveys 45 (3) (2013) 1–35.
[29] R. Engels, Planning tasks for KDD; performing task-oriented user-guidance., in: Proceedings of the International Conference on Knowledge Discovery and Data Mining, KDD '96, 1996, pp. 170–175.
[30] J. Raes, Inside two commercially available statistical expert systems, Statistics and Computing 2 (2) (1992) 55–62.
[31] D. H. Sleeman, M. Rissakis, S. Craw, N. Graner, S. Sharma, Consultant-2: pre- and post-processing of ML applications, International Journal of Human-Computer Studies 43 (1) (1995) 43–63.
[32] P. Brazdil, C. Giraud-Carrier, C. Soares, R. Vilalta, Metalearning: Applications to Data Mining, 1st Edition, Springer Publishing Company, Incorporated, 2008.
[33] A. Kalousis, Algorithm selection via meta-learning. university of geneve, 2002, ph.D. Dissertation.
URL `http://cui.unige.ch/~kalousis/papers/metalearning/PhdThesisKalousis.pdf`
[34] C. Giraud-Carrier, The data mining advisor: meta-learning at the service of practitioners, ICMLA '05, 2005.
[35] M.-A. Zller, M. F. Huber, Survey on Automated Machine Learning, in: arXiv: `https://arxiv.org/pdf/1904.12054.pdf`, 2019.
[36] G. Lindner, R. Studer, AST: support for algorithm selection with a CBR approach, in: Proceedings of the International Conference on Principles of Data Mining and Knowledge Discovery, PKDD '99, 1999, pp. 418–423.
[37] K. Morik, M. Scholz, The miningmart approach, in: Informatik bewegt: Informatik, 2002, pp. 811–818.
[38] C. Diamantini, D. Potena, E. Storti, Ontology-driven KDD process composition, in: Proceedings of the Internations Symposium on Intelligent Data Analysis, IDA '09, 2009, pp. 285–296.
[39] M. Záková, P. Kremen, F. Zelezný, N. Lavrac, Automating KD workflow composition through ontology-based planning, IEEE Transactions on Automation Science and Engineering 8 (2) (2011) 253–264.
[40] J. Kietz, F. Serban, S. Fischer, A. Bernstein, Semantics Inside! But Let's Not Tell the Data Miners: Intelligent Support for Data Mining, in: Proceedings of the International Conference on Extended Semantic Web Conference, ESWC '14, 2014, pp. 706–720.
[41] C. Thornton, F. Hutter, H. H. Hoos, et al., Auto-weka: Combined selection and hyperparameter optimization of classification algorithms, in: KDD, 2013, pp. 847–855.
[42] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, in: Advances in Neural Information Processing Systems, NIPS '15, 2015, pp. 2962–2970.
[43] R. S. Olson, N. Bartley, R. J. Urbanowicz, J. H. Moore, Evaluation of a tree-based pipeline optimization tool for automating data science, in: Proceedings of the International Conference on Genetic and Evolutionary Computation Conference, GECCO '16, 2016, pp. 485–492.
[44] M. Feurer, J. T. Springenberg, F. Hutter, Initializing bayesian hyperparameter optimization via meta-learning, in: Proceedings of the Conference on Artificial Intelligence, AAAI '15, 2015, pp. 1128–1135.

[45] M. Reif, F. Shafait, M. Goldstein, T. Breuel, A. Dengel, Automatic classifier selection for non-experts, Pattern Anal. Appl. 17 (1) (2014) 83–96.
[46] P. Nguyen, M. Hilario, A. Kalousis, Using meta-mining to support data mining workflow planning and optimization, J. Artif. Intell. Res. 51 (2014) 605–644.
[47] B. Bilalli, A. Abelló, T. Aluja-Banet, R. Wrembel, Automated data pre-processing via meta-learning, in: Proceedings of the International Conference on Model and Data Engineering, MEDI '16, 2016, pp. 194–208.
[48] B. Bilalli, A. Abelló, T. Aluja-Banet, R. Wrembel, Intelligent assistance for data pre-processing, Computer Standards & Interfaces 57 (2018) 101–109.
[49] B. Bilalli, A. Abelló, T. Aluja-Banet, R. Wrembel, Towards intelligent data analysis: The metadata challenge, in: Proceedings of the International Conference on Internet of Things and Big Data, IOTBD '16, 2016, pp. 331–338.
[50] K. Alexandros, H. Melanie, Model selection via meta-learning: A comparative study, International Journal on Artificial Intelligence Tools 10 (04) (2001) 525–554.
[51] B. Pfahringer, H. Bensusan, C. G. Giraud-Carrier, Meta-learning by landmarking various learning algorithms, in: Proceedings of the International Conference on Machine Learning, ICML '00, 2000, pp. 743–750.
[52] Y. Peng, P. A. Flach, C. Soares, P. Brazdil, Improved dataset characterisation for meta-learning, in: Proceedings of the International Conference on Discovery Science, 2002, pp. 141–152.
[53] B. Bilalli, A. Abelló, T. Aluja-Banet, On the predictive power of meta-features in OpenML, Applied Mathematics & Computer Science 28 (4) (2017) 697–712.
[54] M. Christ, A. W. Kempa-Liehr, M. Feindt, Distributed and parallel time series feature extraction for industrial big data applications, CoRR abs/1610.07717.
URL `http://arxiv.org/abs/1610.07717`
[55] D. J. Hand, Measuring classifier performance: a coherent alternative to the area under the ROC curve, Machine Learning 77 (1) (2009) 103–123.
[56] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI '95, 1995, pp. 1137–1143.
[57] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, Information Processing and Management 45 (4) (2009) 427–437.
[58] B. Bilalli, A. Abelló, T. Aluja-Banet, R. F. Munir, R. Wrembel, PRESISTANT: data pre-processing assistant, in: Proceedings of the International Conference on Advanced Information Systems Engineering - CAiSE Forum 2018, 2018, pp. 57–65.
[59] L. Breiman, Random forests, Machine Learning 45 (1) (2001) 5–32.
[60] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, 2016, pp. 785–794.
[61] K. Järvelin, J. Kekäläinen, Ir evaluation methods for retrieving highly relevant documents, in: Proceedings of the International Conference on Research and Development in Information Retrieval, SIGIR '00, 2000, pp. 41–48.
[62] H. Steck, Evaluation of recommendations: Rating-prediction and ranking, in: Proceedings of the ACM Conference on Recommender Systems, RecSys '13, 2013, pp. 213–220.
[63] J. Carver, M. L. Jaccheri, S. Morasca, F. Shull, Using empirical studies during software courses, in: Empirical Methods and Studies in Software Engineering, Experiences from ESERNET, Vol. 2765, Springer, 2003, pp. 81–103.